Title: Demonstrate basic concepts of object oriented programming for a class to store details of student.

Objectives: 1) To understand use of different types of constructor and destructor.
2) To understand use of static members and inline keyword.
3) To understand this pointer, dynamic memory allocation operators like new and delete.

Problem Statement: Develop an object oriented program in C++ to create a database of student information system containing the following information: Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving license no. etc Construct the database with suitable member functions for initializing and destroying the data viz constructor, default constructor, Copy constructor, destructor, static member functions, class, this pointer, inline code and dynamic memory allocation operators-new and delete

Outcomes: 1) Students will be able to demonstrate different types of constructor and destructor.
2) Students will be able to demonstrate use of static members and inline keyword.
3) Students will be able to demonstrate this pointer, dynamic memory allocation operators like new and delete.

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: 64 bit Linux/Windows Operating System, G++ compiler

Theory :

Constructor: Constructor is a special member function whose task is to initialize the objects of it's class. This is the first method that is run when an instance of a type is created. A constructor is invoked whenever an object of its associated class is created. If a class contains a constructor, then an object created by that class will be initialized automatically. We pass data to the constructor by enclosing it in the parentheses following the class name when creating an object

Characteristics of a constructor:

1. They should be declared in the public section
2. Constructor name and class name must ne same
3. They are invoked automatically when the objects are created.
4. They cannot be inherited
5. They cannot be virtual

Constructors are special member functions which are having some different properties than that of the normal member functions of C++ class. They are special because they shares exactly same name as that of class name and they gets called automatically when the object of corresponding class is created in memory. In other words it means that constructor is invoked whenever the object of associated class is created. It's called constructor because it construct the values of data members of class.

A constructor is declared and defined as :

```
class Sample
{
```

```
int  a, b;
public:
Sample();
};
Sample :: Sample ()
{
a=0;  b=0;
}
```

When a class contains a constructor like the defined one above, it is guaranteed that an object created by class will initialized automatically

For example:

```
Sample S;
```

It will not only creates the object S of type Sample but also initializes its data members a & b to zero. A constructor with no parameter is called as the default constructor. If no constructor is defined then the complier supplies a default constructor.

The constructors function has some special characteristics:

1. The constructor should be declared in public section.
2. They are automatically invoke when the objects are created.
3. They do not have return type, not even void type is present. They do not have return values.
4. They cannot be inherited though a derived class can call the base class constructor
5. Similar to other C++ function they can have also default argument.
6. Constructors cannot be Virtual.
7. We cannot refer to their addresses.
8. As object with constructor cannot be used as a member of union
9. They may implicit calls to the operators new and delete when memory allocation is required.

Types of Constructors:

Constructors are of three types:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

Default Constructor: Default constructor is the constructor which doesn't take any argument. It has no parameter.

```
class Cube
{
int side;
public:
Cube()
{
side=10;
```

```
}
};
int main()
{
Cube c;
cout << c.side;
}
```

Parameterized Constructor: These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

```
class Cube
{
int side;
public:
Cube(int x)
{
side=x;
}
};
int main()
{
Cube c1(10);
Cube c2(20);
Cube c3(30);
cout << c1.side;
cout << c2.side;
cout << c3.side;
}
```

Copy Constructor: These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

```
class fun
{
float x,y;
public:
fun (float a,float b)//constructor
{
x = a;
y = b;
}
fun (fun &f) //copy constructor
{
```

```
cout<<"\ncopy constructor at work\n";
x = f.x;
y = f.y;
}
void display (void)
{
cout<<""<<y<<end1;
}
};
```

Destructors: Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.
The syntax for destructor is same as that for the constructor; the class name is used for the name of destructor; with a tilde ~ sign as prefix to it.

```
class A
{
public:
~A();
};
```

Destructors will never have any arguments.

Allocation of memory: There are two ways that memory gets allocated for data storage:

- Compile Time (or static) Allocation: Memory for named variables is allocated by the compiler. Exact size and type of storage must be known at compile time. For standard array declarations, this is why the size has to be constant.
- Dynamic Memory Allocation: Memory allocated "on the fly" during run time dynamically allocated space usually placed in a program segment known as the heap or the free store. Exact amount of space or number of items does not have to be known by the compiler in advance. For dynamic memory allocation, pointers are crucial.

Need of Dynamic memory allocation : All memory needs were determined before program execution by defining the variables needed. But there may be cases where the memory needs of a program can only be determined during runtime.
For example, when the memory needed depends on user input. On these cases, programs need to dynamically allocate memory, for which the C++ language integrates the operators new and delete.
Steps for dynamic memory allocation:

1. Creating the dynamic space.
2. Storing its address in a pointer (so that the space can be accessed)
3. Allocating space with new:

To allocate space dynamically, use the unary operator new, followed by the type being allocated.

```
new int;        // dynamically allocates an int
new double;     // dynamically allocates a double
```

If creating an array dynamically, use the same form, but put brackets with a size after the type:

```
new int[40];      // dynamically allocates an array of 40 ints
new double[size]; // dynamically allocates an array of size doubles
//  note that the size can be a variable
```

These statements above are not very useful by themselves, because the allocated spaces have no names! BUT, the new operator returns the starting address of the allocated space, and this address can be stored in a pointer:

```
int * p;        // declare a pointer p
p = new int;    // dynamically allocate an int and load address into p
double * d;     // declare a pointer d
d = new double; // dynamically allocate a double and load address into d
// we can also do these in single line statements
int x = 40;
int * list = new int[x];
float * numbers = new float[x+10];
```

"this" pointer : The 'this' pointer is passed as a hidden argument to all non static member function calls and is available as a local variable within the body of all nonstatic functions. 'this' pointer is a constant pointer that holds the memory address of the current object. 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name).
Following is an example situation where 'this' pointer is used:
When local variable's name is same as member's name

```
#include<iostream>
using namespace std;
/* local variable is same as a member's name */
class Test
{
private:
int x;
public:
void setX (int x)
{
// The 'this' pointer is used to retrieve the object's x hidden by the local
variable 'x'
this->x = x;
}
void print() { cout << "x = " << x << endl; }
};
int main()
{
Test obj;
```

```
int x = 20;
obj.setX(x);
obj.print();
return 0;
}
```

Algorithm:

Flowchart:

Test Cases:

Test Case :

**********Student Information System**********
How many student you have?
1
*******Enter Details of Student ********
Name : hi
Roll No. : 1
Class : se
Div : a
Address: pune
Phone : 123
Date of Birth : 123
Driving License : 123
Blood Group : a

**********Student Information System**********
**********Details of Student 1**********
Name : hi Roll No. : 1
Class : se Div : a
Address: pune Phone : 123
Date of Birth : 123 Driving License : 123 Blood Group : a
**********Details of Student 2**********
Name : hi Roll No. : 1
Class : se Div : a
Address: pune Phone : 123
Date of Birth : 123 Driving License : 123 Blood Group : a

Thank you for deleting details
Thank you for deleting details

**********Student Information System**********
How many student you have?
1

*******Enter Details of Student ********
Name : abc
Roll No. : sd
Class : Div : Address: Phone : Date of Birth : Driving License : Blood Group :
**********Student Information System**********
**********Details of Student 1**********
Name : abc Roll No. : 0
Class : Div :
Address: Phone : 0
Date of Birth : Driving License : Blood Group :
**********Details of Student 2**********
Name : First Roll No. : 2
Class : SE Div : A
Address: Pune Phone : 0
Date of Birth : 10-07-2016 Driving License : 777 Blood Group : B+

Thank you for deleting details
Thank you for deleting details
Conclusion: Hence, we are able to study, use arithmetic operators with switch statement.

PROGRAM

```
#include<iostream>
 #include<string.h>
 #include<stdlib.h>
 using namespace std;
 class Student
 {
 private:
 char *name,*class1,*div,*dob,*bldgrp, *address,*drv;
 int rno,phone;
 static int counter;
 public:
 Student();
 Student(char *,int,char *,int,char *,char *,char *,char *,char *);
 ~Student();
 inline void getDetails();
 inline void showDetails();
 static void count();
 };
 int Student::counter;
 Student::Student()
 {
 name=new char[1];
 class1=new char[1];
 div=new char[1];
 dob=new char[1];
 bldgrp=new char[1];
 address=new char[1];
 drv=new char[1];
```

```
 rno=0;
 phone=0;
}
 Student::Student(char *name,int rno, char *address,int phone,char *class1,
char *div,char *dob,char *bldgrp,char *drv)
 {
 int len=strlen(name);
 this->name=new char[len+1];
 strcpy(this->name,name);

len=strlen(class1);
 this->class1=new char[len+1];
 strcpy(this->class1,class1);

len=strlen(div);
 this->div=new char[len+1];
 strcpy(this->div,div);

len=strlen(dob);
 this->dob=new char[len+1];
 strcpy(this->dob,dob);

len=strlen(bldgrp);
 this->bldgrp=new char[len+1];
 strcpy(this->bldgrp,bldgrp);

len=strlen(address);
 this->address=new char[len+1];
 strcpy(this->address,address);

len=strlen(drv);
 this->drv=new char[len+1];
 strcpy(this->drv,drv);

this->rno=rno;
 this->phone=0;

}

Student::~Student()
 {
 delete name;
 delete address;
 delete dob;
 delete drv;
 delete bldgrp;
 delete class1;
 delete div;
 rno=0;
 phone=0;
 cout<<"\nThank you for deleting details\n";

}
 void Student::getDetails()
 {
 cout<<"\n*******Enter Details of Student ********\n";
 cout<<"Name : ";
```

```
 cin>>name;
 cout<<"Roll No. : ";
 cin>>rno;
 cout<<"Class : ";
 cin>>class1;
 cout<<"Div : ";
 cin>>div;
 cout<<"Address: ";
 cin>>address;
 cout<<"Phone : ";
 cin>>phone;
 cout<<"Date of Birth : ";
 cin>>dob;
 cout<<"Driving License : ";
 cin>>drv;
 cout<<"Blood Group : ";
 cin>>bldgrp;

}
 void Student::showDetails()
 {
 Student::count();
 cout<<"\nName : "<<name;
 cout<<"\tRoll No. : "<<rno;
 cout<<"\nClass : "<<class1;
 cout<<"\tDiv : "<<div;
 cout<<"\nAddress: "<<address;
 cout<<"\tPhone : "<<phone;
 cout<<"\nDate of Birth : "<<dob;
 cout<<"\tDriving License : "<<drv;
 cout<<"\tBlood Group : "<<bldgrp<<endl;
 }
 void Student::count()
 {
 counter++;
 cout<<"\n***************Details of Student
"<<counter<<"***************\n";

}
 int main()
 {
 cout<<"\n**************Student Information System**********\n";
 cout<<"How many student you have?";
 int num;
 cin>>num;
 Student s[num];
 for(int i=0;i<num;i++)
 {
 s[i].getDetails();
 }
 system("clear");
 cout<<"\n**************Student Information System**********\n";
 for(int i=0;i<num;i++)
 {
 s[i].showDetails();
 }
 Student s1("First",2,"Pune",12345,"SE","A","10-07-2016","B+","777");
```

```
 s1.showDetails();

return 0;
 }
```

**********Output **********

How many Students you have?
2
*******Enter Details of Student ********
Name : hi
Roll No. : 1
Class : se
Div : a
Address: pune
Phone : 123
Date of Birth : 123
Driving License : 123
Blood Group : a
*******Enter Details of Student ********
Name : bye
Roll No. : 2
Class : se
Div : a
Address: pune
Phone : 234
Date of Birth : 222
Driving License : 222
Blood Group : b
**********Student Information System**********
**********Details of Student 1**********
Name : hi Roll No. : 1
Class : se Div : a
Address: pune Phone : 123
Date of Birth : 123 Driving License : 123 Blood Group : a
**********Details of Student 2**********
Name : bye Roll No. : 2
Class : se Div : a
Address: pune Phone : 234
Date of Birth : 222 Driving License : 222 Blood Group : b
**********Details of Student 3**********
Name : First Roll No. : 2
Class : SE Div : A
Address: Pune Phone : 0
Date of Birth : 10-07-2016 Driving License : 777 Blood Group : B+